BRASSBOARD DEMONSTRATION OF AN ADVANCED PROCESSOR TECHNOLOGY FOR MULTISENSOR DISCRIMINATING INTERCEPTOR SEEKER

James A. Fabunmi
AEDAR Corporation, Landover MD

Abstract

The data throughput and algorithm intensity requirements of multisensor discriminating interceptor seekers call for new processor technologies which are massively parallel, compact and low power. The Extended Logic Intelligent Processing System (ELIPS) is an advanced processor technology which is being developed to meet these challenges. The ELIPS seamlessly integrates four massively parallel modules into a programmable architecture. These modules are derived from the four main paradigms of artificial intelligence: Feed-Forward Neural Networks, Recurrent Neural Networks, Fuzzy Logic and Expert Systems. These paradigms not only generalize existing signal processing mapping commonly used in digital signal processors, but also introduce additional mappings which capture signal processing concepts directly. This results in significant reductions in algorithm overhead as well as increased parallelism. The processing tasks of multisensor discriminating seekers such as: Target Acquisition, Multitarget Tracking, Feature Extraction, Target Discrimination, Aim Point Selection etc., are formulated as combinations of these functions, operating on data from sensors, memory and/or communications. This paper presents the principles of operation of the ELIPS along with the processor brassboard design. The brassboard implementation utilizes a configurable computing platform which consists of field programmable gate array processing elements and associated interfaces for programmed control by a host processor.

Introduction

The critical processing functions of the Discriminating Interceptor Technology Program (DITP)[1] requires processor architectures which are radically different from any that presently exist. The development of an interceptor which will autonomously discriminate lethal targets from decoys and various other penetration aids, is not a simple matter of upgrading the components of a non discriminating interceptor. The transition from non-discrimination to discrimination capabilities imposes new standards for processing throughput as well as the mix of algorithms which are required to perform the high frame rate image understanding necessary for discrimination functions. The Extended Logic Intelligent Processing System (ELIPS) is being developed in response to the challenges posed by the requirements of the discriminating interceptor.

The ELIPS is an advanced processor technology for on-board interceptor sensor fusion and discrimination. The hardware architecture of the ELIPS will be based on low-power, low-weight compact VLSI-CMOS electronics. The system is an electronically reconfigurable advanced seeker processor, capable of implementing massively parallel algorithms for multisensor detection, tracking, discrimination, feature extraction, situation awareness and sensor management. ELIPS algorithms will seamlessly integrate fuzzy logic, neural network and expert system paradigms. The ELIPS hardware design is modular, such that stacks of ELIPS multichip modules can be assembled as a co-processor to a host CPU.

The ELIPS is implementable in three different technologies, based on the application. ELIPS/SW is a software implementation of the ELIPS using high level programming language such as C, C++, Matlab, FORTRAN etc. All the ELIPS functionalities are implemented as function calls or subroutines. This implementation is suitable for exploratory development of ELIPS approaches to a given application. When the specialized ELIPS functions are implemented using dedicated digital co-processors (e.g. FPGA boards), we obtain the ELIPS/DF (digital firmware). A significant speed up in the execution times is achieved by the ELIPS/DF over the ELIPS/SW. Maximum execution speeds as well as minimum power consumption and minimum processor size is achievable by implementing the ELIPS in a hybrid (analog/digital) architecture. The resulting implementation is the ELIPS/HF (hybrid firmware). This paper addresses the design and demonstration of the ELIPS/DF which is a brassboard for the hybrid, compact, low-power ELIPS/HF. The principles of operation of the ELIPS is first presented, followed by descriptions of the digital designs for the computational engines of the ELIPS modules. An assessment of the implementation of one of the ELIPS modules, the programmable feedforward neural network (PFN) using a commercially available virtual computer board which is based on the Xilinx 6000 series FPGA is then discussed. Preliminary tests already demonstrate a 40-fold increase in processing speed for the implementation of a 64x64 synapse array, when compared to Pentium CPU.

Principles of Operation of the ELIPS

Computational Mappings

In a most general sense, signal processing is the mapping of a set of measured numbers on to another set of numbers which represent derived information based on the measured numbers. The intermediate processes within this overall mapping, are themselves also mappings of numbers onto numbers. The reasoning behind a given signal process is usually captured by a system of equations. If the output numbers are related to the input numbers via some functional relationship, we can consider such a process to be a functional mapping. Functional mappings can be linear or non-linear. Linear functional mappings are represented by equations wherein the output numbers are linear combinations of the input numbers. Examples are transforms, convolutions and matrix multiplications. When viewed this way, it can be seen that today's computational engines, including digital signal processors (DSP), are really only meant for linear mappings. Because of the huge success of these engines, fueled by the technology of VLSI electronics, most signal processing tasks are first reduced to series of linear mappings, which are subsequently coded to run on DSPs.

When the output numbers in a functional mapping cannot be represented by a linear combination of the input numbers, we have what can be considered a non-linear functional mapping. Technically speaking, if the functional relationship between the output and input is known explicitly, such a functional mapping can be approximated by a series of linear mappings. This is in fact what we do when we program today's computers to execute algorithms for effecting non linear functional mappings. What if the functional relationship is not known explicitly (i.e. only exemplar input-output relationships are available)? How do we capture such mappings? What if the functional relationship is known explicitly, but it is inefficient for the mapping to be approximated by linear mappings? How do we accomplish this? This is where artificial neural networks have come to the rescue. Theory holds that multilayer feedforward artificial neural networks can capture the mapping between input and output, so long as sufficient training exemplars are available.

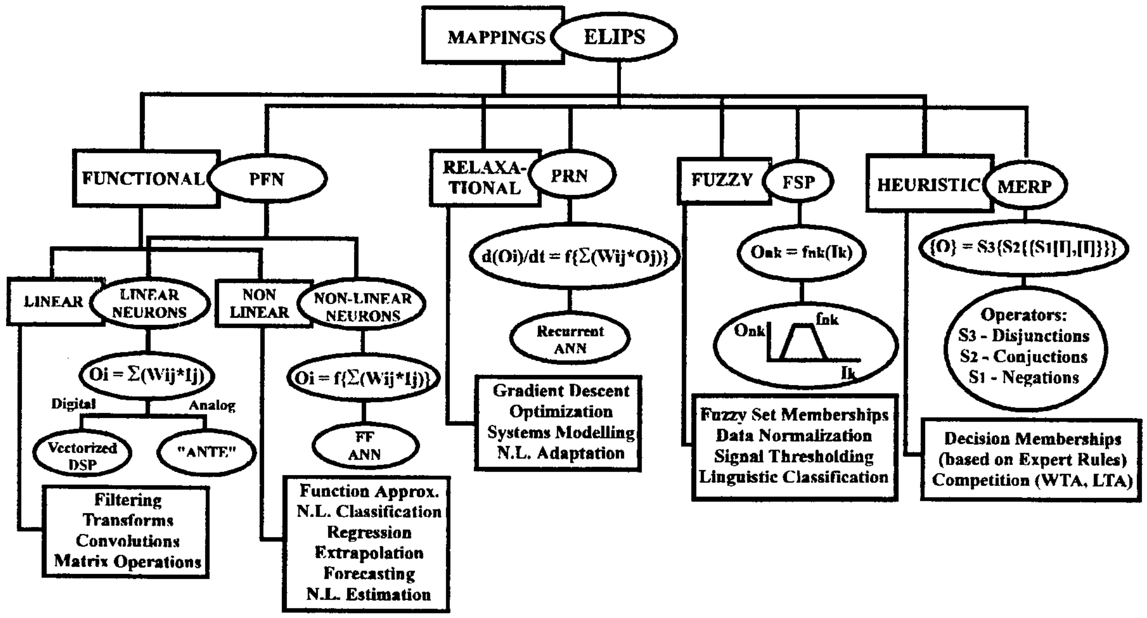
There are signal processing tasks where the input numbers are merely initial states of a process governed by systems of temporal differential equations. The output numbers in such cases are the final states of the relaxation of the dynamical process. For want of a better terminology, lets refer to these mappings as "relaxational" mappings. Generally the systems of temporal differential equations can be cast in to a form wherein the instantaneous rate of change of the state vector is a function of the most recent values of the state vector. In other words, the rate of change of the state vector is a functional mapping of the most recent values of the state vector itself. If this functional mapping is a linear mapping, we have a familiar state-space representation of the dynamical process. A more general situation arises when the functional mapping is non linear.

Besides functional mappings and relaxational mappings, there can also be fuzzy mappings and heuristic mappings. Fuzzy mappings are used to derive memberships of the measured numbers in fuzzy sets defined by heuristic notions of what is being measured. For example, if the measured numbers represent temperature readings from a thermocouple, it may be of interest to assign membership values to fuzzy sets of cold, warm or hot objects. Subsequent signal processing could then be based on reasonings about these linguistic classes. Again, using today's computational engines, this mapping can be accomplished by approximation via series of linear mappings. Heuristic mappings are used to map input facts (coded into numbers) on to output conclusions (also coded into numbers) via sets of expert rules. Linear mappings are useless against these types of signal processes. On today's computers, heuristic mappings usually involve the generation of instructions based on high-level computer languages. The involvement of high level programming language has been a major obstacle in the development of signal processors that could incorporate expert reasoning. On today's computational engines, the most successful approaches to the employment of expert rules in signal processing, has been via the fuzzy logic paradigm.

The ELIPS technology is a bold attempt to capture the mappings of any signal processing task as efficiently as possible by employing computational engines which are best suited to the task. The ELIPS consists of four computational engines - Programmable Feed-forward Neural-network (PFN); Programmable Recurrent Neural-network (PRN), Fuzzy Set Processor (FSP) and the Multistage Expert Rules Processor (MERP). Fig. 1 illustrates the correspondence of the ELIPS engines to the mappings involved in signal processing. The PFN engine is programmable to perform both linear and non linear functional mappings. As a linear functional mapping engine, the PFN is a massively parallel digital signal processor. Such engines are most efficient when utilized to capture such signal processing concepts as Filtering, Transforms, Convolutions and Matrix Operations. A point of reference is to note that the linear digital PFN is equivalent to an array of what is today known as Digital Signal Processors (DSP). The "ANTE" portion of BMDO's "VIGILANTE" program is equivalent to a 3-D implementation of an analog linear PFN. As a non-linear functional mapping engine, the PFN is a multistage feed forward artificial neural network. Examples of signal processing concepts which could be captured directly by the non linear PFN are Function Approximations, Non Linear Classification, Regression, Extrapolations, Forecasting and Non Linear Estimation. In this regard, the PFN alone represents a significant advancement in the development of computational engines.

The PRN engine is programmable to perform relaxational mappings. Its structure is similar to that of recurrent artificial neural networks. Its programmability allows the feedback of the network to be either a linear or non linear functional mapping. Search algorithms involving Gradient Descent, Optimizations and Non Linear Adaptation can be programmed directly using the PRN. In addition, the PRN could be programmed to emulate dynamical systems whose equations have been cast in the state-space form. This type of processing is very useful for optimal control of unknown or poorly defined dynamical systems. The FSP is programmable to perform fuzzy mappings. Natural signal processing performed by humans involves the manipulation and reasoning with sets of data which are referenced by linguistic terms, e.g. temperature could be processed as cold, warm, hot etc. Although the equations for accomplishing these linguistic fuzzy mappings can be ultimately coded into DSP processes, the FSP provides a computational engine which accomplishes these fuzzy mappings directly. Input sensor measurements are immediately mapped on to fuzzy set membership functions. In addition to fuzzy set mappings, the FSP is also a generalized way of performing such operations as Data Normalization, Signal Thresholding and Linguistic Classification.

Finally, the MERP is programmable to perform heuristic mappings based on expert rules. When an expert derives a certain conclusion from a set of facts, the expert has examined a finite set of rules or conditions which are necessary for reaching the chosen conclusion. The expert rules could be provided in a variety of forms, including Rule Trees, If-Then statements etc. On a conventional computer, higher level languages have been developed to translate the expert rules in to forms which allow the computer to process the mapping of input facts on to output conclusions. Because of the need to first interpret these high-level languages before the computational engine could be engaged, it has been difficult to incorporate expert reasoning in algorithms designed for real time, high speed image understanding.



NOTES:
(1) DIGITAL LINEAR PFN COMPUTATIONAL MAPPING IS EQUIVALENT TO THAT OF A VECTORIZED DSP
(2) ELIPS COMPUTATIONAL MAPPINGS CAPTURE SIGNAL PROCESSING AT CONCEPTUAL LEVEL
(3) ALGORITHMIC OVERHEAD IS KEPT TO MINIMUM BY EMPOYING ELIPS MAPPING ENGINES

Fig. 1 ELIPS Computational Mappings for Signal Processing

Digital Engines

The terminology of "engines" in a computational context is used to represent functional modules which are used to accomplish mappings of input numbers to output numbers at the hardware level. Digital engines are composed exclusively of digital logic blocks which operate on digitized input numbers to produce digitized output numbers. The input numbers into an engine can be the signal data or configuration data. Signal data are numbers which are part of the stream of data under process. Configuration data are numbers which are used to define the mathematical function that is being performed within the engine.

Linear Mapping Engine (LME)

The linear mapping engine has two input ports and one output port (see Fig. 2). The two input ports feed numbers into a multiplier block and then into an accumulator. The accumulator adds the result of the multiplication to the most recent value in its register.

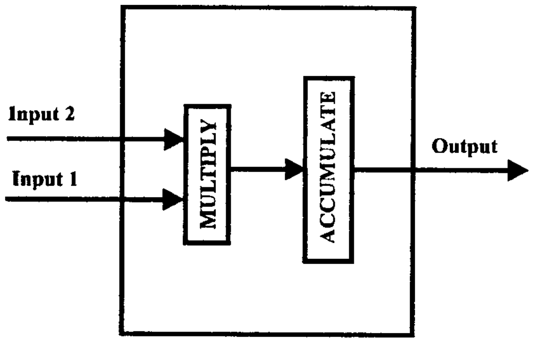


Fig. 2 The Linear Mapping Engine (LME)

The Linear Mapping Engine (LME) is the fundamental engine suitable for performing computations such as matrix multiplication, filtering, transforms and convolutions. An illustration of how an array of LME are used to perform a matrix-vector multiplication is shown in Fig. 3.

Non-Linear Mapping Engine (NME)

The non-linear mapping engine (NME), like the LME also has two input ports and one output

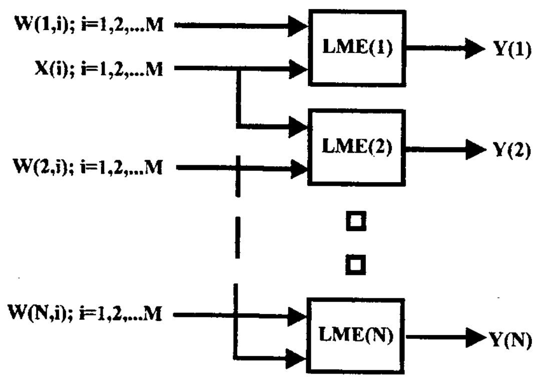


Fig. 3 Parallel Matrix-Vector Multiplication: {Y}=[W]{X}.

port (see Fig. 4). The NME generalizes the LME by inserting a look up table (LUT) between the output of the accumulator and the final output of the mapping engine.

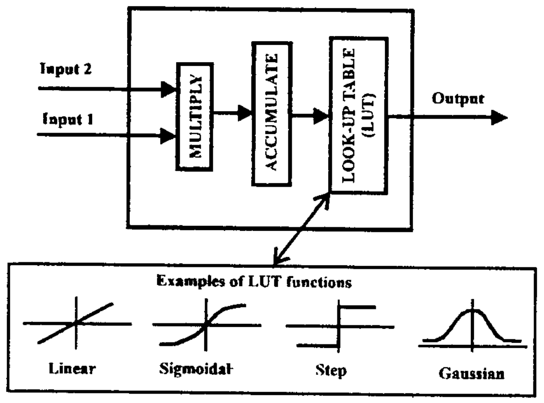


Fig. 4 The Non-Linear Mapping Engine (NME)

The generalization of the NME over the LME is clear from the fact that if the LUT is programmed to be linear, the NME is reduced to an LME. By programming different types of LUT, the NME becomes the fundamental computational engine needed for implementing any feed-forward neural network. An illustration of how an array of NME are used to implement a typical layer of a feed forward neural network is shown in Fig. 5. Theory holds that a multilayer feed-forward neural network can always be trained to emulate any arbitrary functional mapping, provided that a sufficient number of neurons and synapses are available for the adaptation process. This implies that a succession of arrays of NME can be adapted to represent the functional mapping of any set of input numbers on to the required set of output numbers.

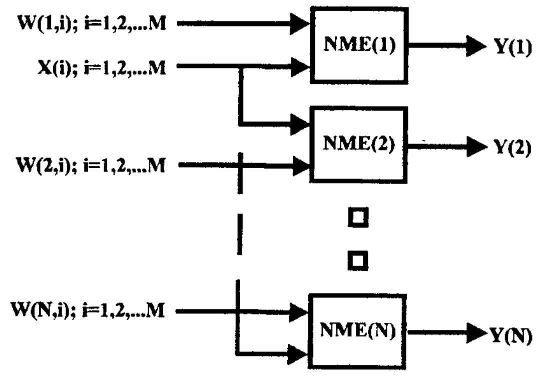


Fig. 5 Typical Stage of a Feed-Forward Neural Network: {Y}=f([W]{X}).

Fuzzy Set Mapping Engine

The fuzzy set mapping engine (FME) has five input ports, and one output port. One of the input ports feeds in the sensor measurement, the other four input ports feed in the parameters of the fuzzy set membership. The output port produces a fuzzy set membership according to the following equations:

$$V(i)=\begin{cases} 0; & X(s(i))\le M(i,1) \\ \dfrac{X(s(i))-M(i,1)}{M(i,2)-M(i,1)}; & \begin{cases} M(i,1)<X(s(i))<M(i,2) \\ M(i,1)\ne M(i,2) \end{cases} \\ 1-\dfrac{X(s(i))-M(i,3)}{M(i,4)-M(i,3)}; & \begin{cases} M(i,3)<X(s(i))<M(i,4) \\ M(i,3)\ne M(i,4) \end{cases} \\ 1; & M(i,3)\ge X(s(i))\ge M(i,2) \\ 0; & X(s(i))\ge M(i,4) \end{cases}$$

The functional relationships required by these equations are used to generate look up tables in order to expedite the operation of the engine (see Fig. 6). A graphical representation of the mapping performed by the FME is illustrated in Fig. 7.

Heuristic Mapping Engine (HME)

The heuristic mapping engine (HME) is illustrated in Fig. 8. The HME has five input ports and two output ports. One of the input ports is used to feed the array of numbers which represent the fuzzy set membership values for a particular sensor. Two other input ports carry encoded rules which are used to select operands for the conjunctive and disjunctive operations.

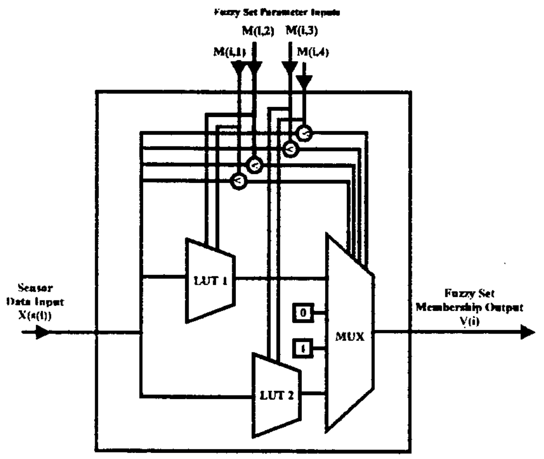

Fuzzy Set Parameter Inputs
M(i,2)
M(i,3)
M(i,1)
M(i,4)
Sensor Data Input X(s(i))
LUT 1
LUT 2
0
1
MUX
Fuzzy Set Membership Output V(i)

Fig. 6 Fuzzy Set Mapping Engine (FME)

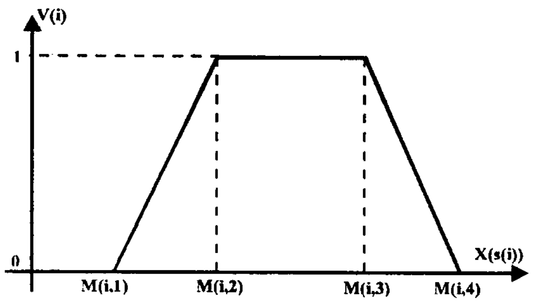


Fig. 7 Graphical Illustration of FME Mapping with Linear Look Up Tables.

The remaining two-input ports carry residual conjunctions and disjunctions from preceding HME in architectures where multiple HME are connected. The two output ports are used to pass forward the results of conjunctive and disjunctive operations performed by the HME. In the most general case, an expert rule can always be reduced to a sequence of disjunctive and conjunctive operations, acting over a set of input facts. For example, the degree of truth of the jth conclusion D(j) can be computed as:

D(j) = max(min(j,1),min(j,2), ...min(j,Nj))

where.

min(j,k) = min(fact(j,k,1), fact(j,k,2), ...fact(j,k,Mjk))

and fact(j,k,m) is the mth fact membership operand of the kth conjunctive operand in the jth disjunction which defines the value of D(j). The selection of the fact membership operands as well as the conjunctive operands are encoded into binary streams which are used to select the correct operand for a given expert rule. Considering that a fact membership appearing in a particular expert rule can itself be the negation of an input measured fact membership, it is necessary to compute an array of numbers corresponding to the negative memberships of each measured fact membership. This negative membership is computed as the maximum value of the array of fuzzy set memberships measured from the same sensor, excluding the fact membership in question. The physical meaning of this calculation is as follows: Suppose the measured data from a given sensor is assigned memberships in fuzzy sets s(j), j=1,2,...N. The fact of a measurement belonging to set s(k) is established to a degree given by the fuzzy set membership. On the other had, the fact of a measurement not belonging to set s(k) means that the measurement could belong to any other set - s(1) or s(2) or ....s(n), excluding s(k).

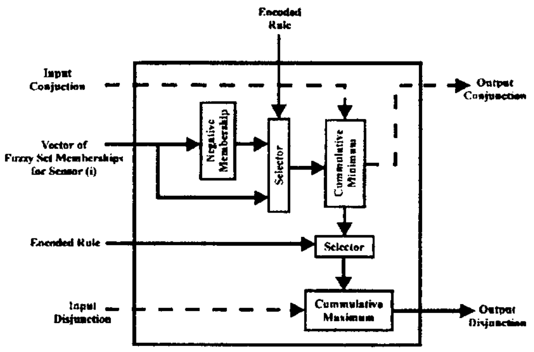


Fig. 8 Heuristic Mapping Engine (HME)

ELIPS Brassboard (ELIPS/DF)

The ELIPS/DF is based on commercially available virtual computer boards consisting of FPGA processing elements and PCI interface with the host computer. Fig. 9 and Fig. 10 are pictures of boards marketed by Giga Operations Inc, and Virtual Computer Corporation respectively.

PFN/DF Architecture

The programmable feed-forward neural network (PFN) is the component of the ELIPS which is designed to perform functional mappings. The computational engine which drives the PFN is the non-linear mapping engine (NME) shown in Fig. 4. The PFN/DF can be realized on a Reconfigurable Computer (RC) via the implementation of three functions, callable by software (e.g. C, C++) running on the host CPU.

Fig. 9 The G900 Reconfigurable Interface Card.

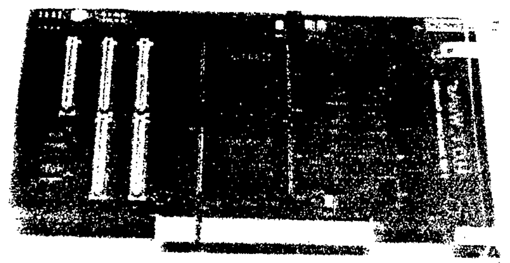


Fig. 10 The PCI-XC6200 Virtual Computing Development System.

1. Function PFNSTP(ns, NI(ns x 1), NO(ns x 1), NLF(ns x 1)) configures the RC with predefined registers (for storage of weights) and arithmetic blocks (bitstream adders and multipliers and look up tables for neuronal transfer functions), as well as addresses for the passing Input/Output (I/O) data. (ns) is the number of stages of the PFN/DF, NI is a ns-element vector of the dimensions of the input data into each stage, NO is a ns-element vector of the dimensions of the output data out of each stage. NLF is a ns-element vector of the indices specifying the transfer functions of the neurons of the ns-stage of PFNSTP. There should be a check that the number of outputs of a given stage should match the number of inputs of the succeeding stage. The result of invoking this function is the reconfiguration of the RC logic blocks, interconnections and registers. As a practical issue, configurations of the RC for preset values of the parameters ns, NI and NO would have been stored in some RAM. When a run time call for PFNSTP is made, a determination of the preset values which are available, and large enough to accommodate the run time call is made. That means that the actual reconfiguration will be done with preset parameters which are at least as large as the run-time values. Fig. 11 illustrates the stages of the PFN. Each stage is similar to the architecture shown in Fig.5.

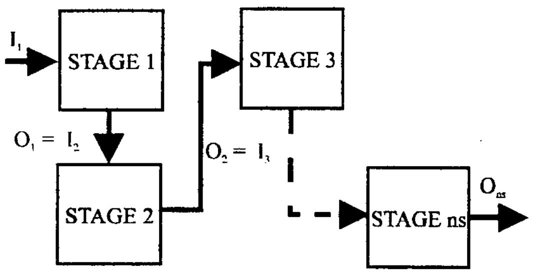


Fig. 11 Multiple Stages of the PFN/DF

2. Function PFNCFG(WTS(nwts x 1)) configures the RC by loading values of weights into the appropriate registers as defined by PFNSTP. WTS is the array of weights; nwts is the size of the array of weights. For a PFN with no bias weights, the value of nwts is determined by the sum of the product of input and output for all the stages of the PFN. Again we note that the actual values will correspond to the preset architecture from memory which accommodates the run-time values. The registers for the unused weights are defaulted to zeros.

3. Function PFNRUN(I,O) accomplishes the actual data processing using the RC. The parameters I, O specify the addresses of the data which are input and output respectively. In other words, external input data is applied at the addresses specified by the parameter I. The output of PFN/DF is passed to the address locations specified by the parameter O.

PRN/DF Architecture

The programmable recurrent neural network (PRN) is the component of the ELIPS which is designed to perform relaxational mappings The implementation of the PRN is analogous to that of the single-stage PFN. The main difference is that in the PRN, there is feedback of the neuronal outputs in to the input, with a unit delay (see Fig. 12).

FSP/DF Architecture

The fuzzy set processor is the component of the ELIPS which is designed to perform fuzzy mappings. The computational engine which drives the FSP is the fuzzy set mapping engine (FME) shown in Fig. 6. The digital firmware implementation of the FSP is done on a reconfigurable computer (RC) per the following specifications: The RC board interfaces with a PCI slot in a desk top personal computer. The functions to be performed by the RC board are seamlessly callable from a C++ and/or Matlab programming environments. FSPSTP(I,C) is a call to configure the RC to perform the fuzzy mapping process. FSPCFG(M) is a function call to specify the elements of the matrix M. Output = FSPRUN(Input) shall be a call to compute the Ouput vector of dimension (nf x 1), given an Input vector of dimension (I x 1).

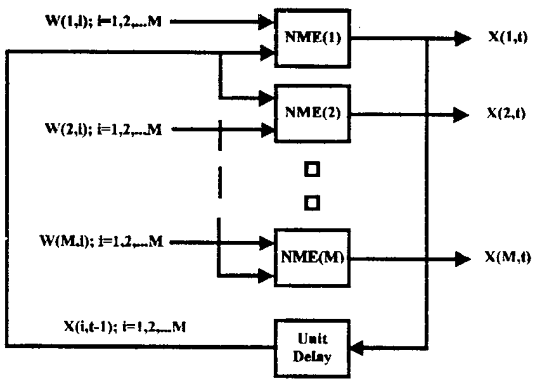


Fig. 12 Architecture of the PRN/DF

MERP/DF Architecture

The multistage expert rules processor (MERP) is the component of the ELIPS which is designed to perform heuristic mappings. The computational engine which drives the MERP is the heuristic mapping engine (HME) shown in Fig. 8. The MERP/DF is implemented on a reconfigurable computer using arrays of heuristic mapping engines, combined to implement the encoded expert rules (see Fig. 13). Depending on the computational resources available on the chip, several HME could be connected in parallel. Otherwise, each HME would be used to process multiple inputs.

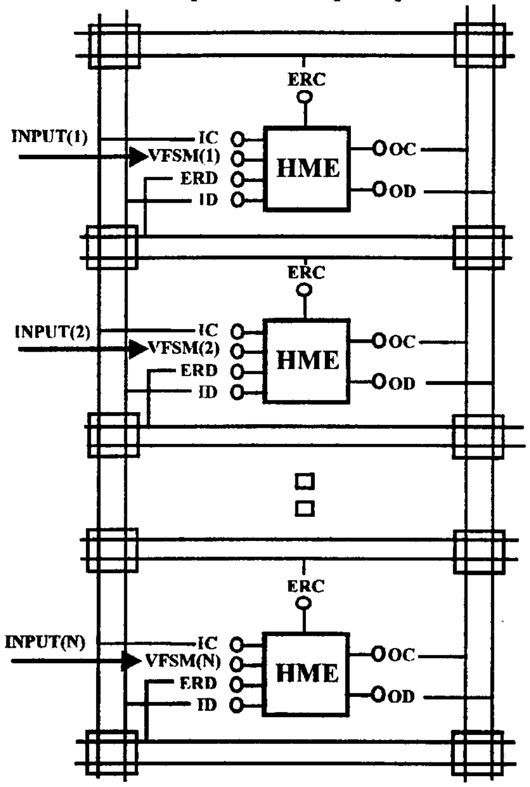


Fig. 13 Architecture of the MERP/DF

Programming and Adaptation

There are two distinct aspects of programming the ELIPS processor. The first aspect is the determination of the sequence and selection of mappings required to accomplish a given signal process. The outcome of this step is the specification of mapping stages and the sequence of those stages. A signal process may require any and all combinations of the four mapping stages of the ELIPS. The same mapping stage may appear multiple times with different configurations. Some mapping stages may be omitted altogether. This aspect of the programming is intimately connected with the physical concepts underlying the signal process. The ELIPS signal processing approach encourages massively parallel formulation of the signal process. The way to maximize the effectiveness of the ELIPS is to parallelize the problem as much as possible, and then select the minimal sequence of the mapping stages needed. to yield the desired outcome.

The second aspect of programming the ELIPS processor is the determination of the configuration parameters required for each mapping stage. Depending on the nature of the signal process, these parameters could be specified explicitly or implicitly. Explicit specification simply implies the creation of appropriate data files with the numbers corresponding to the required parameters. Implicit specification involves an adaptive process, in which the ELIPS is "trained" to create those parameters. This is a powerful aspect of the ELIPS in that if a mapping is required for which only examples of such mappings are known, and not the parameters of the mapping, a training algorithm can be folded around the ELIPS to "teach" it the proper parameters for such a mapping. This is typically what is called for when the PFN is being used to approximate a non-linear function. The known function is used to generate training data (input/output pairs). The mapping parameters are then adjusted using a suitable adaptation scheme, in order to match the exemplar training data. Once a satisfactory set of parameters have been obtained, they can be saved in a configuration file which is subsequently invoked in order to engage the ELIPS signal process. It should be noted that nothing prevents the adaptations scheme from being implemented using ELIPS mapping stages. As a matter of fact, this is strongly encouraged, since then one would end up with a self contained ELIPS system which could be reprogrammed on-line.

ELIPS Program Development Environment

The program development environment required by the ELIPS processor must facilitate the two aspects of ELIPS programming as discussed in the preceding paragraphs. It turns out that both the ELIPS/SW and ELIPS/DF are ideal for this. Since both implementations of the ELIPS operate in conjunction with a host digital processor, any additional resources needed from the host environment could be readily accessed. The ELIPS/SW is implementable on any desktop computing environment, and is suitable for scaled down exploration of the signal process in order to confirm the selection and sequence of the mapping stages required by the process. If the target ELIPS system is an ELIPS/HF, then the programming should include simulations of the hardware constraints such as bit resolution, noise, pin numbers etc.

Preliminary Results

The ELIPS approach offers three levels of opportunity for improving the speed and data throughput of the signal processor. First, by providing appropriate computational engines for capturing the signal process directly, it eliminates a considerable amount of algorithm overhead which would otherwise be unavoidable using conventional processors. Secondly, each module of the ELIPS has a massively parallel computational architecture. Thirdly, the hybrid implementation of the ELIPS will perform the arithmetic operations of multiplication and addition using analog components. This offers a significant increase in computational speed, when compared to digital approaches. The results obtained so far have confirmed the first level.

Using the PCI-XC6200 virtual computer board, a 64x64 PFN/DF was implemented. Because of the limited computational resources on the chip, the parallelism was limited. It was only possible to input two pairs of 14-bit numbers at a time into the multiply/accumulate cycle. The look up table for the neuronal transfer function was done using the on-board SRAM. With all these limitations, the FPGA board was still 40 times faster than the host pentium CPU, in performing the same 64x64 synapse single stage PFN/DF process.

Acknowledgments

This effort was supported in part by a contract funded by the Ballistic Missile Defense Organization (BMDO). The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation herein.

References

1. Figie, B., Kinashi, Y., Johnson, B., Linderman, R., Fabunmi, J., 1996, "Discriminating Interceptor Technology Program (DITP): Sensor Fusion for Improved Interceptor Seekers", 1996 AIAA/BMDO Missile Sciences Conference, Monterey CA.